SEMI-ANALYTIC SOLUTIONS TO THE RADIATIVE TRANSFER EQUATIONS VIA HETEROGENEOUS COMPUTING

A Thesis

by

DANIEL ALPHIN HOLLADAY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ryan McClarren
Committee Members,	Marvin Adams
	Nancy Amato
	John Wöhlbier
Head of Department,	Yassin Hassan

December 2014

Major Subject: Nuclear Engineering

ABSTRACT

High energy density radiative transfer benchmark solutions are presented for a 1-D slab geometry using a three-temperature (electron, ion, and radiation) model and 1-D spherical geometry using a two-temperature (material, radiation) model. A transport model is used for the radiation, a conduction model is used for the electrons, and ion and/or material motion is assumed negligible. These benchmarks are useful in the verification and testing of simulation codes for laboratory astrophysics as well as high energy density physics (HEDP). The solutions require linearization of the coupled equations and are obtained via specific cubic functional forms (in temperatures) for the heat capacities and electron-ion coupling factor. These solutions are semi-analytic in that their exact forms can be written down, but 2-D integrals must be computed numerically for each point in space and time. These integrals are slowly convergent and so a numerical integration routine was developed in OpenCL to take advantage of the high throughput that heterogeneous computing offers. Although capable of running on any OpenCL device, the nature of numerical integration meant GPUs were an excellent choice. Using a figure of merit analogous to flops per watt, the OpenCL implementation achieves 25x better performance with respect to this figure of merit, and an overall speedup of 560x was observed over a serialized implementation of the same algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

If fusion energy is to be harnessed on earth, a thorough understanding of the regime called high energy density physics must be very well understood. Due to Planck's law, when temperatures in the high energy density range are attained in a physical system, energy in the form of thermal photons will make up a significant amount of the total system energy in addition to the normal internal energy and kinetic energy. The equations governing this exchange of energy between the radiation "field" and the material energy are called the radiative transfer equations. These equations are important when it comes to modeling plasmas to compute reaction rates for many different thermonuclear processes such as inertial confinement fusion.

There are several large scale computer codes such as xRage developed at Los Alamos National Laboratory (LANL), KULL developed at Lawrence Livermore National Laboratory (LLNL), and Hydra also developed at LLNL; these codes numerically solve the radiative transfer equations by discretizing time and space. These codes are constantly under modification and development. Unit testing is implemented to verify that the codes work properly after changes are made. Analytic solutions are an excellent tool for code verification because the true error obtained by the code is calculable. Tests will verify that the codes still get the right answer and approach the right answer at the correct rate. As computer architectures continue to improve in both speed and efficiency, these software packages are able to include higher fidelity physical models allowing for much more physically correct simulations. These constant changes and improvements require not only unit testing of individual functions, but integrated testing to test the package in a much more inclusive way, making sure that the independent components that make up the soft-

ware package are working together correctly. The two models analyzed were the 2 temperature (2-T) and 3 temperature (3-T) models for thermal radiative transfer. The 3-T model allows for the electron and ion energy fields to be out of equilibrium with each other, while the 2-T model assumes the electron and ion species are in thermal equilibrium. The 3-T can be useful useful when attempting to model physical phenomena such as the input of energy from a laser, which preferentially heats electrons, and hydrodynamic shock waves, which preferentially heat ions. To ensure that these models are being solved correctly, analytic solutions were are sought.

Analytic solutions for both the 2-T and 3-T models are presented, where the 3-T model uses a 1-D slab radiation source and the 2-T model using a 1-D spherical radiation source. The spherical source is especially useful because it can be used to test the 3-D functionality of codes such as xRage and KULL. To compute the solutions, the material property dependences were chosen to be certain cubic polynomials with respect to the material temperature(s) such that the equations could become linear partial differential equations (PDEs). The solution method involved using spatial Fourier and temporal Laplace transforms. The inverse transforms were written in the form of double integrals over either infinite or semi-infinite domains in wavenumber frequency $(k-\omega)$ space. These integrals were computed numerically at specific points and space and time. For every point in space-time, a double integral was computed numerically. A composite Gauss-Legendre quadrature rule was used to compute the integrals. This rule was implemented via OpenCL and computed on 4 Nvidia Tesla graphics processing units (GPUs).

2. THE EQUATIONS OF THERMAL RADIATIVE TRANSFER

The radiative transfer equations describe the transport of radiative energy in a physical system and the exchange of radiative energy with its environment. The radiation transport equation in it's most general form is given by:

$$\frac{1}{c}\frac{\partial I\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)}{\partial t}+\nabla\cdot\left(\mathbf{\Omega}I\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)\right)+\sigma_t\left(\mathbf{r},\nu,t\right)I\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)=\varepsilon\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)+$$
$$\int_0^\infty \mathrm{d}\nu'\int_{4\pi}\mathrm{d}\Omega'\left[\sigma_s\left(\mathbf{r},\nu'\rightarrow\nu,\mathbf{\Omega}'\cdot\mathbf{\Omega},t\right)I\left(\mathbf{r},\nu',\mathbf{\Omega}',t\right)\right]+Q\left(\mathbf{r},\nu,\mathbf{\Omega},t\right), \qquad (2.1)$$

where c is the speed of light, $I\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)$ is the radiative intensity, $\mathbf{r}$ is a position vector, ν is the photon frequency, $\mathbf{\Omega}$ is a vector on the surface of the unit sphere, t is the time variable, $\sigma_t\left(\mathbf{r},\nu,t\right)$ is the macroscopic total photon interaction cross section, $\varepsilon\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)$ is the total emissivity, $\sigma_s\left(\mathbf{r},\nu'\rightarrow\nu,\mathbf{\Omega}'\cdot\mathbf{\Omega},t\right)$ is the macroscopic double differential scattering cross section, and $Q\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)$ is an extraneous source. Every term in equation (2.4) has units of photon energy path length per unit volume per unit frequency per unit solid angle per unit time.

In many cases the system is in a state called local thermodynamic equilibrium (LTE). Under this assumption, the emission term is given by:

$$\varepsilon\left(\mathbf{r},\nu,\mathbf{\Omega},t\right)=\sigma_a\left(\mathbf{r},\nu,t\right)B\left(\nu,T_e\left(\mathbf{r},t\right)\right), \qquad (2.2)$$

where $\sigma_a\left(\mathbf{r},\nu,t\right)$ is the macroscopic photon absorption cross section and $B\left(\nu,T_e\left(\mathbf{r},t\right)\right)$

is the Planck distribution. It is given by:

$$B(\nu, T) = \frac{2h\nu^3}{c^2} \frac{1}{\exp\left(\frac{h\nu}{k_B T}\right) - 1}, \tag{2.3}$$

where h is Planck's constant and k_B is the Boltzmann constant. Many such analyses integrate the radiation transport equation over all photon frequencies to simplify the analysis. This is called the 1-group, or gray, approximation and under this approximation, the emission term simplifies greatly due to the fact that the Planck distribution integrated over all frequencies is analytic. The 1-group radiation transport equation under LTE is given by:

$$\frac{1}{c}\frac{\partial I(\mathbf{r}, \mathbf{\Omega}, t)}{\partial t} + \nabla \cdot (\mathbf{\Omega} I(\mathbf{r}, \mathbf{\Omega}, t)) + \sigma_t(\mathbf{r}, t) I(\mathbf{r}, \mathbf{\Omega}, t) = \sigma_a(\mathbf{r}, t) \frac{acT_e^4(\mathbf{r}, t)}{4\pi} + \int_{4\pi} \mathrm{d}\Omega' \left[\sigma_s(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega}, t) I(\mathbf{r}, \mathbf{\Omega}', t))\right] + Q_r(\mathbf{r}, \mathbf{\Omega}, t), \tag{2.4}$$

where $a \equiv \frac{8\pi^5 k_B^4}{15h^3c^3}$ is the radiation constant. Additional equations are needed to describe the material energy field.

2.1 3-Temperature Model

In the high energy density regime, the material is very likely ionized (i.e. a plasma) and it is likely that the cross section for photo-ion interactions is different than the cross section for free electron interactions which means that their mean energies could differ. In the 3-T model, it is assumed that the radiation, free electron, and ion energy distributions can be described via a temperature. This difference in mean energies would imply that the free electron and ion temperatures can differ. If a heat conduction model is used for electrons and ions, then the electron and ion

energy equations are given by:

$$c_{v,e}(\mathbf{r},t)\frac{\partial T_e(\mathbf{r},t)}{\partial t}-\nabla\cdot\left(\kappa_e(\mathbf{r},t)\nabla T_e(\mathbf{r},t)\right)=c\sigma_a\left(\phi(\mathbf{r},t)-aT_e^4(\mathbf{r},t)\right)+\gamma_{ei}(\mathbf{r},t)\left(T_i(\mathbf{r},t)-T_e(\mathbf{r},t)\right)+Q_e(\mathbf{r},t) \tag{2.5a}$$

$$c_{v,i}(\mathbf{r},t)\frac{\partial T_i(\mathbf{r},t)}{\partial t}-\nabla\cdot\left(\kappa_i(\mathbf{r},t)\nabla T_i(\mathbf{r},t)\right)=\gamma_{ei}(\mathbf{r},t)\left(T_e(\mathbf{r},t)-T_i(\mathbf{r},t)\right)+Q_i(\mathbf{r},t) \tag{2.5b}$$

Equations (2.4), (2.5a), and (2.5b) fully describe the exchange of energy between radiation, ion, and electron energy fields given that the system is in local thermodynamic equilibrium and in a regime such that electron and ion conduction models are accurate, meaning that electron diffusion term is physically accurate. Ion motion occurs on such a slow time scale that it can be neglected in many cases. The benchmark problem being considered does not have electron or ion extraneous sources, and photon scattering is assumed to be negligible. In addition, the benchmark problem is a one dimensional problem in a Cartesian geometry. This yields the following equations:

$$\frac{1}{c}\frac{\partial I(x,\mu,t)}{\partial t}+\mu\frac{\partial}{\partial x}\left(I(x,\mu,t)\right)+\sigma_a(x,t)I(x,\mu,t)=\sigma_a(x,t)\frac{acT_e^4(x,t)}{2}+Q_r(x,\mu,t) \tag{2.6}$$

$$c_{v,e}(x,t)\frac{\partial T_e(x,t)}{\partial t}-\frac{\partial}{\partial x}\left(\kappa_e(x,t)\frac{\partial}{\partial x}\left(T_e(x,t)\right)\right)=c\sigma_a(x,t)\left(\phi(x,t)-aT_e^4(x,t)\right)+\gamma_{ei}(x,t)\left(T_i(x,t)-T_e(x,t)\right) \tag{2.7a}$$

$$c_{v,i}(x,t)\frac{\partial T_i(x,t)}{\partial t}=\gamma_{ei}(x,t)\left(T_e(x,t)-T_i(x,t)\right) \tag{2.7b}$$

2.2 2-Temperature Model

In a 2 temperature model, ions and electrons are assumed to be in thermal equilibrium with each other and thus treated as a single material energy field. The material energy conduction is assumed to be negligible. The correct material energy equation is equivalent to the electron energy equation with $\gamma_{ei} = \kappa_e = 0$ and $T_i = T_e = T$. The radiation transport equation remains the same and is still given by equation (2.6). For the 2-T model, the material energy equation is

$$c_v(x,t)\frac{\partial T(x,t)}{\partial t} = c\sigma_a(x,t)\left(\phi(x,t) - aT^4(x,t)\right) \tag{2.8}$$

Solutions to the above equations, given certain functional forms for the radiation, ion, and electron energy source terms are sought.

2.3 Previous Benchmark Solution Work

The radiative transfer equations are highly studied and as such, previous work has been accomplished in the realm of benchmark solutions. Most notably, Su and Olson published benchmark solutions for radiation transport under a two temperature (2-T) model, in which ions and electrons are assumed to be in equilibrium with each other and the material energy field can be treated as a single unit [1]. Other work has focused on radiative diffusion with a 3-T model [2].

In these papers, high accuracy tables were presented that were a result of numerically computing an integral for each value in the table. In these cases, integral convergence was obtained via math libraries or programs such as `Mathematica`. These methods were initially attempted to compute the integrals presented in this work, but integral error estimates were high and the computing the integrals to tighter error bounds required unacceptable amounts of time. Many runs were killed after

more than 1.5 weeks of running. Thus, it was highly time prohibitive to use these methods and so a new approach was taken. Much of this work has been to develop a numerical integrator specifically optimized for these types of problems, even with this optimization, highly parallel and advanced architectures were still needed. The solution method is outlined in the next section.

2.4 Solution Method

2.4.1 Equation Linearization

By making certain assumptions about the physical properties of the material, the equations can be linearized. If the specific heat capacities and the electron-ion coupling constant are proportional to cubic polynomials in both temperatures, the equations can be linearly recast with respect to T_α^4 where $\alpha = e, i$. A set of linear partial differential equations can become a set of linear equations via Fourier and Laplace transforms. One can then solve for the transformed variables and invert the transforms. The inverse transforms of these equations can be written down but not computed analytically. They involve computing difficult integrals numerically, and thus the solutions are known as "semi-analytic" solutions as the solution can be written down but not computed analytically [1]. The forms for these properties are shown in the non-dimensionalization section.

2.4.2 Non-Dimensionalization

By introducing the following parameters, the equations of thermal radiative transfer can be written in a non-dimensional form using the following:

[1] This is analogous to the function $\exp(x)$ being the "semi-analytic" solution to the differential equation $\frac{\mathrm{d}y}{\mathrm{d}x} = y$.

$$\tau = c\sigma_a t, \qquad z = \sigma_a x, \qquad \hat{T}_\alpha = \frac{T_\alpha}{T_H}, \qquad c_{v,\alpha} = 4aT_\alpha^3, \qquad \gamma_{ei} = \sigma_a acT_H^3 \frac{\hat{T}_i^4 - \hat{T}_e^4}{\hat{T}_i - \hat{T}_e}\hat{\gamma},$$
$$\kappa_\alpha = 4aT_\alpha^3 D_\alpha, \quad \hat{\kappa}_\alpha = \frac{\sigma_a D_\alpha}{c}, \quad w = \frac{\phi}{acT_H^4}, \quad u = \frac{I}{acT_H^4}, \qquad v_\alpha = \hat{T}_\alpha^4,$$

where α can be either e for the electronic species material properties or i for the ionic species material properties. The non-dimensional and linearized 3-T equations of thermal radiative transfer in a scatter free medium are given by:

$$\frac{\partial u}{\partial \tau} + \mu \frac{\partial u}{\partial z} + u = \frac{v_e}{2} + \tilde{S}_r, \tag{2.9a}$$
$$\frac{\partial v_e}{\partial \tau} + v_e = \hat{\kappa}_e \frac{\partial^2 v_e}{\partial z^2} + w + \hat{\gamma}\left(v_i - v_e\right), \tag{2.9b}$$
$$\frac{\partial v_i}{\partial \tau} = \hat{\gamma}\left(v_e - v_i\right). \tag{2.9c}$$

2.4.3 Integral Transforms

Using Fourier transforms in space and Laplace transforms in time, the above system of coupled linear PDEs is transformed into a linear system of 3 equations.

$$\left(s + \mu ik + 1\right)\mathcal{U} = \frac{\mathcal{V}_e}{2} + \mathcal{S}_r, \tag{2.10a}$$
$$\left(s + 1\right)\mathcal{V}_e = -k^2\hat{\kappa}_e\mathcal{V}_e + \mathcal{W} + \hat{\gamma}\left(\mathcal{V}_i - \mathcal{V}_e\right), \tag{2.10b}$$
$$s\mathcal{V}_i = \hat{\gamma}\left(\mathcal{V}_e - \mathcal{V}_i\right), \tag{2.10c}$$

where calligraphed characters represent the doubly transformed quantities:

$$\mathcal{F}\left(k, s\right) = \int_0^\infty \mathrm{d}\tau \int_{-\infty}^{\infty} \mathrm{d}z f\left(z, \tau\right) \exp\left(-\left(ikz + s\tau\right)\right). \tag{2.11}$$

The radiation transport equation can be integrated over angle to obtain the angle integrated photon intensity in terms of other quantities:

$$\mathcal{W} = \left(\frac{\mathcal{V}_e}{2} + \mathcal{S}_r \right) b\left(k, s\right), \tag{2.12}$$

where $b\left(k, s\right)$ is defined by:

$$b\left(k, s\right) \equiv \int_{-1}^{1} \frac{1}{s + \mu i k + 1} \mathrm{d}\mu. \tag{2.13}$$

This essentially eliminates equation (2.10a), thus the system becomes a 2 variable system of equations (2.10b) and (2.10c). Solving for the doubly transformed quantities is simple and the solution is given below:

$$\mathcal{W} = \left(\frac{\mathcal{V}_e}{2} + \mathcal{S}_r \right) b, \tag{2.14a}$$

$$\mathcal{V}_e = \frac{b\mathcal{S}_r \left(s + \hat{\gamma}\right)}{\left(1 - \frac{b}{2} + s + \hat{\gamma} + k^2\kappa_e\right)\left(s + \hat{\gamma}\right) - \hat{\gamma}^2}, \tag{2.14b}$$

$$\mathcal{V}_i = \frac{b\mathcal{S}_r \hat{\gamma}}{\left(1 - \frac{b}{2} + s + \hat{\gamma} + k^2\kappa_e\right)\left(s + \hat{\gamma}\right) - \hat{\gamma}^2}. \tag{2.14c}$$

Equations (2.14a) - (2.14c) are the exact solutions to this problem in wavenumber – frequency $(k - s)$ space. By allowing $\hat{\gamma} \to 0$ and $\kappa_e \to 0$, the two temperature solutions can be recovered. They need to be transformed back into the non-dimensional space – time $(z - \tau)$ space. To do this, the inverse Fourier and Mellin integral transforms are applied:

$$f\left(z, \tau\right) = -\frac{i}{(2\pi)^2} \oint_{\text{Bromwich}} \mathrm{d}s \int_{-\infty}^{+\infty} \mathcal{F}\left(k, s\right) \exp\left(s\tau + ikz\right) \mathrm{d}k \tag{2.15}$$

Here, the Bromwich contour is a semicircular contour in the complex plane that encloses all of the poles of $\mathcal{F}$ and can be seen illustrated in Figure 2.1. It can be shown that all of the poles are to the left of the imaginary axis [3] which allows for the straight portion of the contour to lie on the imaginary axis. This contour integral can be broken up into 2 pieces, one over the imaginary axis and one over the semi-circular arc. It can be shown that the integral over the semi-circular arc is 0. The Mellin integral transform can be rewritten as an integral over the imaginary axis.

$$-\frac{i}{2\pi}\oint_{\text{Bromwich}} \mathrm{d}s = -\frac{i}{2\pi}\left[\int_{\gamma-i\infty}^{\gamma+i\infty} \mathrm{d}s + \cancel{\int_{\text{arc}} \mathrm{d}s}\right] \tag{2.16}$$

In this case, $\gamma = 0$ and thus the substitution $s \to i\omega$ can be made which will flip the integration to along the real axis instead of the imaginary axis:

$$-\frac{i}{2\pi}\oint_{\text{Bromwich}} \mathrm{d}s \to \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{d}\omega \tag{2.17}$$

This leads to a form of equation (2.15) that is more amenable to numerical integration.

$$f(z,\tau) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty} \mathrm{d}\omega \int_{-\infty}^{+\infty} \mathcal{F}(k,\omega)\exp\left(i\left(\omega\tau + kz\right)\right)\mathrm{d}k \tag{2.18}$$

In order to evaluate these solutions at different spatial and temporal locations, the integrals must be computed numerically, which is discussed in detail in the next section.

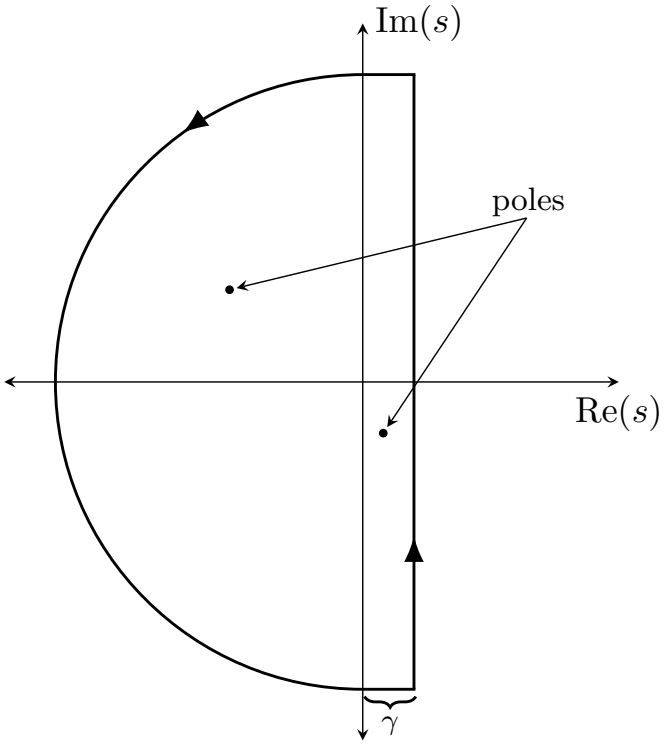

Figure 2.1: Bromwich contour must enclose all of the poles of the function, if all of the poles are to the left of the imaginary axis, the straight part of the contour lies on the imaginary axis.

3. NUMERICAL INTEGRATION STRATEGIES

In order to compute the semi-analytic solutions to the radiative transfer equations numerical integrals must be computed. There are several methods for estimating a definite integral, and the method of composite Gauss-Legendre quadrature was the method of choice.

3.1 Gauss-Legendre Quadrature

Composite Gauss-Legendre quadrature is a composition of Gauss-Legendre quadratures. The composition will be described later in this section. First, Gauss-Legendre quadrature will be discussed. In general, a quadrature is a method of approximating an integral in the following way:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i), \tag{3.1}$$

where $f(x)$ is the integrand, w_i is the set of quadrature weights, x_i is the set of abscissae, and n is the approximation order. Together, $\{w_i,\ x_i\}$ make up the quadrature set. The abscissae for a Gauss-Legendre are the zeros of the n^{th} order Legendre polynomial, $\{x_i | P_n(z_i) = 0,\ x_i \in [-1,1]\}$. This quadrature is chosen to utilize every degree of freedom to integrate polynomials of maximum degree, in this case $2n-1$. This quadrature set exactly integrates the Lagrange interpolation of $f(x)$ interpolated at the quadrature points:

$$f(x) \approx \sum_{i=1}^{n} f(x_i) \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \tag{3.2}$$

The definition of the weights arrises from integrating the above interpolant

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} f(x_i) \underbrace{\int_{-1}^{1} \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}\mathrm{d}x}_{w_i}. \tag{3.3}$$

The Gauss-Legendre weights given by the following formula:

$$w_i = \frac{2\left(1 - x_i^2\right)}{(n+1)^2 \left[P_{n+1}(x_i)\right]^2}, \tag{3.4}$$

which comes from known properties of the Legendre polynomials. The code developed in this work, named numintCL , recomputes quadrature sets every it is executed. Sets are computed for 2, 4, 8, 16, 32, and 40 point Gauss-Legendre quadratures.

3.1.1 Composite Rule

Gauss-Legendre quadrature has excellent convergence properties as a function of the quadrature order, but the overall integration is also related to the domain size. Additionally, high quadrature order can cause arithmetic precision issues. Increasing the quadrature order reduces the integration error (p-refinement), but one can also split up the domain into subdomains and perform the integration on each subdomain. Splitting the integral into n equally spaced subdomains each of width h over $[a, b]$ yields:

$$\int_{a}^{b} f(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} \int_{a+ih}^{a+(i+1)h} f(x)\,\mathrm{d}x. \tag{3.5}$$

This is called a composite rule and is analogous to h-refinement. Algorithmically, when a refinement occurs, the number of sub-domains is increased by an integer

factor, typically 2 in this work. The quadrature rule needs to be remapped for every subdomain with the following substitution

$$\int_{a+ih}^{a+(i+1)h} f(x)\,\mathrm{d}x = \frac{h}{2}\int_{-1}^{1} f\left(\frac{h}{2}(x'+1)+a+ih\right)\mathrm{d}x'. \tag{3.6}$$

The above substitution is equivalent to mapping to the reference element, thus allowing for the same quadrature set to be used to compute any integral. numintCL starts with 2 subdomains and simultaneously increases the quadrature order and refines the composite rule until convergence or until the maximum quadrature order has been reached, in which case it will only refine the composite rule.

3.1.2 Extension to d-Dimensions

The extension to d dimensional integrals implemented in this work is not applicable to integrals with variable limits of integration, thus restricting integration to hyper rectangular domains. Extending a Gauss-Legendre quadrature rule to d dimensions in theory is not complicated. The integral of a function $f(x_1, x_2, ..., x_d)$ can be approximated via n^{th} order quadrature in much the same way as a 1 dimensional function, except now there is a sum and a weight for every dimension

$$\int_{-1}^{1}\cdots\int_{-1}^{1} f(x_1, ..., x_d)\,\mathrm{d}x_1\cdots\mathrm{d}x_d \approx \sum_{i_1=1}^{n}\cdots\sum_{i_d=1}^{n} f(x_{1,i_1}, ..., x_{d,i_d})\prod_{j=1}^{d} w_{i_j}. \tag{3.7}$$

An equivalent way to look at the above expression is a sum of the function evaluated at n^d quadrature points in a d dimensional volume. To specify a given quadrature point, d numbers (or coordinates) are needed, and each point will have an associated weight. In this modified set, there are n^d points, $\mathbf{x}_i$ each vector of length d and n^d weights, which is merely a product of weights from the original quadrature set. The

above approximation can be converted to a sum that is much less daunting in terms of implementation:

$$\int_{\mathbf{-1}}^{\mathbf{1}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \sum_{i=1}^{n^d} w_i f(\mathbf{x}_i)\,. \tag{3.8}$$

The sum is now fairly simple from an implementation perspective. The previous expression was difficult from an implementation perspective due to the variable number of sums. In a computer code, a sum is typically implemented via a loop, but creating a function that allows for a variable level of loop depth is not trivial, which is why equation (3.8) is a preferable form. Unfortunately, in order to compute the new quadrature set, a similar loop structure is still required. A recursive function was used to achieve this. From a performance perspective, recursive functions are not optimal, but since the quadrature set only needs to be computed once per code execution, this turns out to be a very small portion of the overall integral computation. Applying a composite rule in d dimensions can be implemented on a per dimension basis. Refinements can be made in any number of dimensions independently, and the same substitutions can be made to map from $[a_i, b_i]$ to $[-1, 1]$.

3.1.3 Infinite and Semi-Infinite Domains

The integrals derived in the previous chapter are 2-D integrals over semi-infinite integration domains. The quadrature formulas presented in the previous section are only applicable to finite domains. Two different solutions to this problem were implemented. The two solutions are a variable substitution and iteration. The variable substitution will be outlined, followed by the iterative method.

3.1.3.1 Variable Substitution

Instead of computing an integral over an infinite or semi-infinite domain, one can compute an integral over a finite domain with a modified integrand that yields the same result. Two such methods will be presented, one for infinite domains and one for semi-infinite domains. For infinite domains, allow $x \to t\left(1-t^2\right)^{-1}$, the integral becomes

$$\int_{-\infty}^{\infty} f\left(x\right) \mathrm{d}x = \int_{-1}^{1} f\left(\frac{t}{1-t^2}\right) \frac{1+t^2}{\left(1-t^2\right)^2} \mathrm{d}t. \tag{3.9}$$

Initially, this integral looks very nice especially because it is on the interval $[-1, 1]$, which is the same as for Gauss-Legendre quadrature. However, this nicety does not apply when using a composite rule. If one of the limits of integration is finite (a semi-infinite integration domain), the above substitution does not work. Semi-infinite integration domains can occur when the integrand in equation (2.18) is even with respect to 1 or more integration variables due to symmetry. An integral over such the domain $[0, \infty]$ can be transformed into an integral on $[0, 1]$ by letting $x \to t\left(1-t\right)^{-1}$

$$\int_{0}^{\infty} f\left(x\right) \mathrm{d}x = \int_{0}^{1} f\left(\frac{t}{1-t}\right) \frac{1}{\left(1-t\right)^2} \mathrm{d}t. \tag{3.10}$$

The benefit of these integrals is that the equivalence is exact, no approximations are made. However, the integrands are highly oscillatory and vary greatly in magnitude. A spatially adaptive rule would be highly beneficial for these types of integrands. The method used is not adaptive and thus refines the entire integration domain uniformly, which is potentially highly inefficient.

3.1.3.2 Iteration

The iterative method iterates on what finite upper bounds of integration approximate ∞ with respect to the integral estimate. There is no substitution involved, but it is an approximation. The method works for integrals of the form:

$$\int_0^\infty f(x)\,\mathrm{d}x, \tag{3.11}$$

and in d dimensions. The method could be extended to handle an arbitrary *finite* lower limit of integration, but that is not currently implemented. The method works by integrating over some prescribed domain, increasing the limits of integration, and integrating over the newly introduced domain. Not including the initial domain, there will be $1 + d!$ subdomains to integrate for the integration over the new, larger subdomain is introduced. This is illustrated in 2-D in figure 3.1 in which after integrating over the initial domain, 3 more integrations occur to produce a larger rectangular region. The same treatment can be applied again to create an even larger rectangle, this procedure can be repeated until successive contributions to the overall integral are smaller than the tolerance. Due to the fact that these solutions exist, there is an expectation of convergence for these integrals and the fact that the integrands will approach 0 as $k, \omega \to \infty$. If one of the 3 integral estimates is less than the tolerance consecutively, then when the domain is increased, its corresponding integral will not be computed.

3.2 Acceleration Techniques

Due to the relatively crude implementation of Gauss-Legendre quadrature on uniformly refined hyper rectangular domains, alternative methods of acceleration were sought after. Two acceleration methods were eventually implemented, those

being asymmetric refinement and sequence acceleration. Both of these methods reduce the time to obtain a converged integral and are discussed below.

3.2.1 Asymmetric Refinement

The key to this method of acceleration is to not refine along every dimension uniformly or simultaneously. Refinements occur based on sensitivity of the integral with respect to refinements along a given dimension. Before the refinement begins, a sensitivity analysis is completed to see which dimension causes the greatest change in the integral upon refinement. The dimensions are ordered based upon this sensitivity analysis, $x_1, ..., x_d$. The composite rule is refined along x_1 until the estimate has converged. Once the integral estimate has converged with respect to x_1, all of the other dimensions (i.e. $x_2, ..., x_d$) are refined. One could potentially generalize the prescribed method to refine only in one dimension at a given time and go in order from most to least sensitive. This was not implemented in this work as 2 dimensional integrals were the focus. In this case, there are only 2 levels, an "outer" and an "inner" loop. When the "outer" integral estimate has converged, the integral is considered to be converged.

3.2.2 Error-Based Derefinement

When spending all computational resources to converge the integral estimate with respect to a single dimension, the grid can become more refined than what is needed for a given tolerance. A symptom of this problem is the error estimate for the "inner" refinement loop is many orders of magnitude smaller than the overall integration tolerance. If this occurs, the other dimension will be refined, but the most sensitive dimension will be de-refined by a prescribed number of levels.

3.2.3 Sequence Acceleration

Four different methods of sequence acceleration were also used to accelerate the convergence of integral estimates. As refinements are made, the error in the integral estimates should be reduced. This could be viewed as a convergent sequence of numbers, and there are methods of accelerating convergent sequences. The four methods used in this work are called iterated Aitken δ^2, Wynn ϵ acceleration, Wynn ρ acceleration, and iterated Brezinski θ acceleration.

3.2.3.1 Iterated Aitken δ^2 Acceleration

Before introducing the iterated Aitken δ^2, a brief introduction of the (non-iterated) Aitken δ^2 process will be given. Given three numbers in a converging sequence x_i, x_{i+1}, and x_{i+2}, with x_{i+2} being the newest iterate, and thus the one that is closest to convergence. A better estimate for the converged value, x', is given by

$$x' = x_{i+2} - \frac{(x_{i+2} - x_{i+1})^2}{x_{i+2} - 2x_{i+1} + x_i}. \tag{3.12}$$

Suppose now a list of 5 values in a convergent sequence. One could perform 3 different Aitken δ^2 calculations and obtain 3 accelerated values. These 3 values also belong to a convergent sequence and thus the acceleration method can be performed on these values as well. This technique is called the *iterated* Aitken δ^2 acceleration method [4]. For the n^{th} iteration, the Aitken accelerated values are given by

$$x_i^n = x_{i+2}^{n-1} - \frac{\left(x_{i+2}^{n-1} - x_{i+1}^{n-1}\right)^2}{x_{i+2}^{n-1} - 2x_{i+1}^{n-1} + x_i^{n-1}}. \tag{3.13}$$

This turns out to be the most reliable method for a number of different cases [4, 5] including the integrals computed in this work.

3.2.3.2 Wynn ϵ Acceleration

The Wynn ϵ algorithm is another method of sequence acceleration. The algorithm is based on Padè approximates and its underlying theory is beyond the scope of this work, but can be found in [6]. A convergence table is the output of this algorithm in much the same way as for iterated Aitken δ^2, but every other column of this table is divergent. The large divergences of the even columns of the table means that a high level of precision is required to compute these tables without too significant a loss in numerical accuracy. To compute the i^{th} value of the n^{th} column, the following formula is used

$$x_i^{n+1} = x_{i+1}^{n-1} + \frac{1}{x_{i+1}^n - x_i^n}, \qquad x_i^{-1} = 0. \tag{3.14}$$

This method was also found to be highly reliable for converging these integrals. The above algorithm produces a table of values, but not a square table. Notice to compute a new column with I rows, $I + 1$ rows are needed in its left neighboring column. Additionally, while even columns are filled with convergent numbers, the odd columns are filled with divergent numbers. A new table can be constructed of only the even columns, but as the columns move to the right, every one will have two less rows than its left neighboring column and two more rows than its right neighboring column. This fact is true for every one of the methods explored with the exception of the Brezinski θ algorithm.

3.2.3.3 Wynn ρ Acceleration

The Wynn ρ algorithm is almost identical to the ϵ algorithm, but does not appear to work as well. In general, the ρ algorithm is given by

$$x_i^{n+1} = x_{i+1}^{n-1} + \frac{\rho_{i+n+1} - \rho_i}{x_{i+1}^n - x_i^n}, \qquad x_i^{-1} = 0. \tag{3.15}$$

Using difference choices for ρ leads to different convergence properties. The choice made for this work was that $\rho_\alpha = \alpha + 1$, leading to

$$x_i^{n+1} = x_{i+1}^{n-1} + \frac{i+1}{x_{i+1}^n - x_i^n}, \qquad x_i^{-1} = 0. \tag{3.16}$$

Although not very reliable or as accurate in speeding up the convergence of these integrals, the derivation of this algorithm serves as the basis for Brezinski θ algorithm [7].

3.2.3.4 Iterated Brezinski θ Acceleration

The iterated Brezinski θ algorithm is said to be one of the better and more robust acceleration techniques [7, 4]. However, such positive results have not been seen in this work. To compute the i^{th} value of the n^{th} column, the following formula is used

$$x_i^{n+1} = x_{i+1}^n - \frac{\left(x_{i+1}^n - x_i^n\right)\left(x_{i+2}^n - x_{i+1}^n\right)\left(x_{i+3}^n - 2x_{i+2}^n + x_{i+1}^n\right)}{\left(x_{i+3}^n - x_{i+2}^n\right)\left(x_{i+2}^n - 2x_{i+1}^n + x_i^n\right) - \left(x_{i+1}^n - x_i^n\right)\left(x_{i+3}^n - 2x_{i+2}^n + x_{i+1}^n\right)}. \tag{3.17}$$

Since this method requires 4 values of the sequence, the amount of useful numbers is reduced by 3 when moving from left to right in the columns, instead of 2.

3.2.3.5 Convergence Criteria

Each of these methods produces a table whose values are approaching the true value as one moves to the right and down in the table. This means that the values along the lower diagonal are going to be best possible estimates. To test convergence, all values on the diagonal are compared to every other value on the diagonal, in addition to the row 1 above the diagonal in every column (this row number will be different in every column). If the relative and/or absolute difference between any of those two numbers is less than the relative and absolute tolerances, respectively, the value is said to have converged for a given method to within the tolerances provided. If the method that converged was either Wynn ϵ or Aitken δ^2, then the sequence acceleration method is considered to be converged. In addition, if any 2 or more methods have converged, then the sequence is said to be converged, and the 2 values with the lowest relative difference is the value that is chosen.

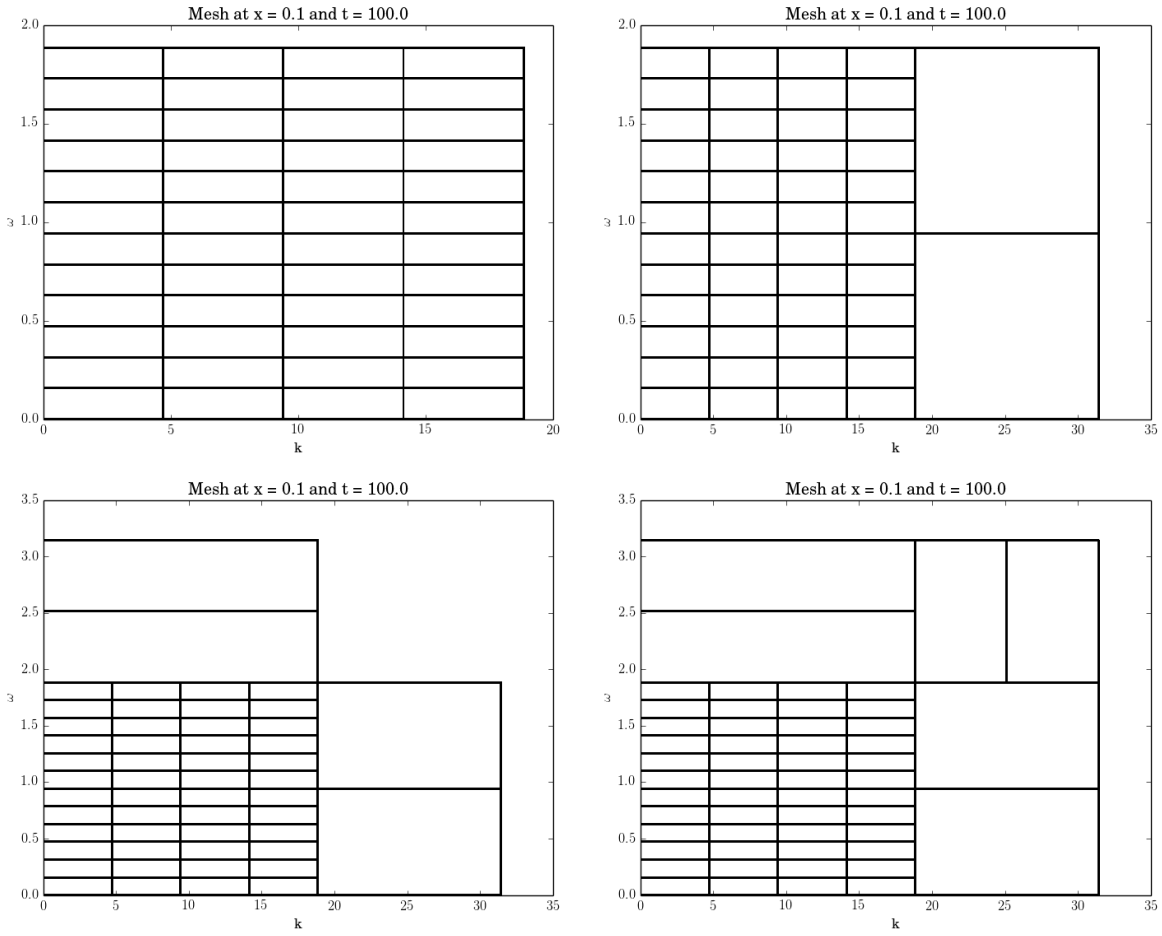

Figure 3.1: Integrating over infinite domains via iteration: After the initial rectangle, an integration over 3 $(= 2! + 1)$ additional rectangular domains are computed in addition, producing a new, larger rectangle. The number of cells along each dimension is equal to the number of refinements that occurred along that dimension.

4. PARALLEL IMPLEMENTATION OF NUMERICAL INTEGRATION WITH OPENCL

As computer interfaces became more complex, the demand for graphics specific capabilities increased and led to the specialized hardware designed specifically for processing graphics. This was called the graphics processing unit and it is made up of hundreds or thousands of low power processing units. These units run at lower clock speeds when compared to central processing units (CPUs) and have access to much less cache per core. Numerical integration of an analytic integrand is perfect for use on GPUs as evaluating the integrand at a point in space requires no inter-processor communication and very little memory. Almost all of the work is in raw computation and not as much in memory motion. To take advantage of the GPUs, the Open Computing Language (OpenCL) was used. OpenCL is an open standard in which hardware vendors can choose to conform to and provide an implementation for their hardware. This allows for specialized hardware and accelerators to be used without the need to use domain specific languages, preprocessor pragmas, or libraries. Software developers can write functions called "kernels" in compliance with the C99 standard, and those kernels can be executed on any OpenCL compatible device. Currently, there are many different types of devices with an OpenCL software development kit (SDK) including multi-core central processing units (CPUs), grahpics processing units (GPUs), and field programmable gate arrays (FPGAs).

4.1 Using OpenCL

The OpenCL is a standard in which hardware vendors can choose to write an SDK to comply with this standard. In the canonical case, there are two pieces of hardware, the host and the device. The host is a standard CPU that can execute

the "host" code. This host code controls kernel executions on the device. The host code includes several calls to the OpenCL application programming interface (API) before actually executing a kernel. First, the OpenCL device(s) must be identified and a context and command queue must be created for the device(s). Once a context and command queue have been created, the kernel(s) must be compiled. OpenCL features runtime compilation of the kernel(s), meaning that they are compiled when the code is executed. Although this could potentially be a hit in overall code performance, it allows for a semblance of metaprogramming due to the fact that runtime parameters in the host code can become compile time parameters for the kernels which can allow optimizations including loop unrolling and the fixing of array sizes. Once the kernel has been compiled, the memory required for the computation needs to be allocated on the device and the data transferred to the device. The data is transferred via peripheral component interconnect express (PCIe); this data transfer is typically the performance bottleneck observed in most heterogenous computation profiling, especially for bandwidth limited computations. Fortunately, numerical integration via Gauss-Legendre quadrature on a uniform grid, if programmed correctly, is compute bound and requires relatively small amounts of memory. Even so, at the beginning of code execution, the GPU and CPU implementation of the integration schemes are profiled to determine the number of subdomains required for the GPU to to have a significant performance improvement over the CPU. If the number of subdomains for a given integration is less than this number, the integral is computed on the CPU, if it is greater, the integral is computed on the GPU. Once the memory has been allocated and the data transferred, the kernel arguments must be set then the kernel can be executed. This execution is controlled by the command queue and can potentially invoke multiple kernels and send and receive data from the host simultaneously. In this case, such concurrency is not necessary. Kernel execution has

its own nomenclature that is specific for OpenCL. When executing a kernel, generally the total problem size is called the global size. The global size can be 1, 2, and 3 dimensional, but since the integrals are in general d dimensional, a one dimensional global size was chosen. The code implementation had that the total problem size, which is the number of subdomains in the composite rule, be at least twice the global size. This is due to the fact that every thread computes the integral over multiple subdomains and is strided by the global size. The total number of subdomains is always chosen to be a power of 2 as the hardware is generally optimized for such sizes. The global size parameter is the product of two parameters called the local size and the number of work groups. In general, the global problem is broken up into a number of work groups, each composed of work items. The number of work items in a work group is called the local size. A work item is the smallest unit of work in the hierarchy. The global size and local size are needed to enqueue a kernel.

4.2 OpenCL Middleware and ocl-mla

The steps and API calls required by most codes utilizing OpenCL are very similar and thus much of the device, context, and command queue declaration has been termed "boilerplate" code. There are now several different versions of middleware available that handles much of the boilerplate code under the hood. One such version is called the OpenCL Middle LAyer, or ocl-mla. This library was developed by Ben Bergen at Los Alamos National Laboratory and is available on github. The code developed in this work uses ocl-mla to handle much of OpenCL API calls under the hood. Additionally, ocl-mla is able to handle the use of multiple devices simultaneously. The use of ocl-mla greatly reduced development time and enabled the easy use of multiple simultaneous OpenCL devices.

4.3 The Multi-Stage Reduction

In order to reduce the amount of information transferred from the device back to the host at the end of the calculation, a device side reduction is used. The type of reduction is the two stage reduction described in [8]. This reduction method parallelizes only enough to match the devices capacity, and within one of these groups the reduction is done in serial. This minimizes the amount of barriers and waiting that occurs across work groups. The maximum value for the local size is hardware dependent but can be queried at run time. In order to ensure that every compute unit had a sufficient amount of work, the following inequality was enforced:

$$\#\text{ of subdomains} \geq 2 \times \text{local size} \times \#\text{ of work groups} \tag{4.1}$$

Additionally, the local size needed to be both as large as possible and a power of 2. The number of workgroups was a free parameter. By varying the number of work groups and comparing the wall clock time for total integration execution, it was determined that for the GPUs used (Nvidia Tesla with the Fermi architecture), 2^{15} work groups yielded the best results. This meant that for every device, 2^{15} numbers had to be transferred back to the host and added up. If a significant number of these values was not a number (NaN), infinity, or denormalized, that would be reflected in the output and left up to the user whether or not he/she wanted to reject the results.

4.3.1 Automatic Domain Decomposition

Since multiple devices were used simultaneously, a method for equally dividing of the work was devised. For a d dimensional domain, there is a list of integration variables $x_1, ..., x_d$. If there were n devices, then the domain of integration for x_d would be split into n subregions and each of the devices would receive a different

portion of the x_d subdomain, with the integration domains in all of the other variables remaining the same. This requires that the number of subdomains for x_d be evenly divisible by the number of devices. The number of devices used was 4 and the number of subdomains was always a power of 2, so this requirement was always satisfied.

4.4 Host Implementation

In addition to the OpenCL and ocl-mla portions of numintCL , a significant amount of development time was spent on the host side. Coding the iterative method for computing integrals over semi-infinite domains required significant, error estimation, and convergence acceleration was not a small part of this work. The flow of the code will now be outlined. numintCL first defines the absolute and relative integration tolerances, number of integration dimensions, and the spatial and temporal points at which the analytic solutions will be evaluated. When the number of integration dimensions is defined, Gauss-Legendre quadrature sets of 2nd, 4th, 8th, 16th, 32nd, and 40th order are computed, but any other list of orders could also be computed up to 40th. Above 40 has not been verified and numerical errors associated with the root finding algorithm could lead to significant numerical integration error. When the quadrature sets have been computed and stored, the serial CPU and OpenCL GPU integration functions are profiled to see at which number of subdomains does the GPU implementation significantly outperform the CPU implementation. Since the number of subdomains in which this performance crossover occurs is dependent upon the complexity of the integrand, so the profiling takes place at runtime. This profiling allows for the GPUs to be used when their computational power is needed and the CPU to be used when the cost of computing the integral estimate is small. The profiling is followed by the computation of the analytic solution at the specified spatial and temporal points. In order to compute the solution at a

single point, a double integral over the entire plane must be computed. However, it turns out that for all of the cases presented, some level of symmetry could be taken advantage of. The integrals were always symmetric with respect to the frequency variable, ω. This meant that the integral over all ω was replaced by an integral over half of the space and a factor of 2. For the cases in which the radiation source was a 1-D slab, there was symmetry in the wavenumber variable, k, as well. This was not the case for the spherical radiation source geometry. The iterative method computes integrals on $(0,\infty)^d$. This works perfectly for the cases in which there is symmetry in both k and ω, but not as well spherical source case. Due to the significant lower time to completion with the iterative method vs the integral transform method, the following manipulation can be made:

$$\int_0^\infty \mathrm{d}\omega \int_{-\infty}^{\infty} f(k,\omega)\,\mathrm{d}k = \int_0^\infty \mathrm{d}\omega \int_{-\infty}^{0} f(k,\omega)\,\mathrm{d}k + \int_0^\infty \mathrm{d}\omega \int_0^{\infty} f(k,\omega)\,\mathrm{d}k \tag{4.2}$$

The second integral is now in a form that the iterative method can handle, and by making the substitution of $k \to -k$, the first term goes to an integral on $(0,\infty)^2$ as well. Thus, to obtain the solution at one point, essentially twice the work needs to be accomplished and more error is introduced, but the values can be calculated with the much faster iterative method. The iterative method works by starting with a hyper rectangular integration domain of the form $\prod_i (0, b_i)$ followed by the integral over the domain $\prod_i (b_i, c_i)$. This does not form a hyper rectangular region with the only exception being $d = 1$. The goal is to compute the integral over the region $\prod_i (0, c_i)$, and so the integral is broken up into $d!$ hyper rectangular domains computing the

equivalent integral of

$$\int_{\mathbf{0}}^{\mathbf{c}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\mathbf{0}}^{\mathbf{b}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\mathbf{b}}^{\mathbf{c}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} + \sum_{i=1}^{d!} \int_{\mathbf{x}_{l,i}}^{\mathbf{x}_{u,i}} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

$$\implies \sum_{i=1}^{d!} \int_{\mathbf{x}_{l,i}}^{\mathbf{x}_{u,i}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\mathbf{0}}^{\mathbf{c}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} - \int_{\mathbf{0}}^{\mathbf{b}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} - \int_{\mathbf{b}}^{\mathbf{c}} f(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{4.3}$$

For illustrative purposes, this is shown graphically for the 3-D case below. First are the 2 initial regions shown in figure 4.1. The following 6 (3!) regions are shown in figure 4.2. As was previously said, the goal was to compute the integral over $\prod_i (0, c_i)$. The process can start again, this time with larger limits of integration, and be repeated until successive integral estimates are less than the tolerance. In addition, the estimate of the integral is changing less and less as the approximation of the true semi-infinite domain becomes better, and thus can be thought of as a convergent sequence and sequence acceleration can be used. This machinery requires the ability converge an integral estimate in a given domain with fixed upper and lower limits of integration. For a given integral, the composite rule starts out with 4 subregions per dimension and a 2nd order Gauss-Legendre quadrature order. Two refinements are made in each dimension with a 2nd order rule while no other refinements are made to the other dimensions so that a sensitivity coefficient can be computed for every dimension. The dimension with the largest sensitivity coefficient will be the first dimension to be refined, however, if two sensitivity coefficients are close to each other, more refinements and a higher order quadrature rule is made create a better approximation for the sensitivity coefficient. At each stage of the sensitivity coefficient analysis, the integral estimates are checked for convergence. If convergence has not yet been established, the refinements will continue. The dimension in which

the integral estimate is the most sensitive is refined until convergence according to the sequence acceleration convergence criteria. Upon convergence of this dimension, every other dimension is refined by one level and based upon the error estimate for the most recent inner integral estimate, the sensitive dimension is refined or de-refined and the process of refining the sensitive dimension starts again. When the outer sequence acceleration convergence criteria is met, the integral in the given region is believed to be converged. To compute these composite quadrature rules with a set number of subdomains, the the integration functions are called (either the CPU or the GPU based upon the number of subdomains). The GPU implementation has to recompute the correct global and local sizes, transfer data, set kernel arguments, invoke the kernel, and retrieve and process the data, and the CPU implementation is a fairly straightforward function call. The above summarizes the flow of numintCL concisely yet with enough detail to understand some of the challenges associated with developing numintCL with the level of generality and speed optimization that was built in.

4.5 Performance Results

There were two methods of measuring performance, the first was overall wall clock time to completion. For the second metric, the number of floating point operations per second (FLOPS) per watt initially sounded like an excellent candidate as it takes into account the fact that the GPUs consumed much more power than the CPU could at any given time. However, when it came to actually computing these numbers, it was quickly discovered that they are not easily obtained. The issue with the number of FLOPS is that these integrands contain a number of special functions and the number of flops and memory accesses to compute a special function like a sine or a cosine changes based upon its argument. Additionally, modern compilers perform

a number of optimizations to these integrands and it can be difficult to know what actually gets compiled, especially on the OpenCL side of things because the device code that is compiled all occurs on the device. Instead, one might think to look at a polynomial with only multiplication, addition, and subtraction, and to deactivate any compiler optimizations that might occur. Although this could yield FLOPS for polynomial evaluation, there many other steps in computing the integral. Instead, an analogous metric was used, and that was the number of integrand evaluations per second. Additionally, in order to obtain the average power consumed during a calculation, the power consumption of the devices and only the devices as a function of time would need to be known. Instead of actual power, the maximum thermal design power (TDP) was used for this comparison. Thus, the figure of merit (FOM) used in this performance comparison is given by

$$\smile \equiv \frac{\#\text{ of integrand evaluations}}{\text{t} \times P_{TDP}}, \tag{4.4}$$

where t is the total wall clock time of the calculation and P_{TDP} is the thermal design power of the device(s). For the CPU, the implementation was not parallel and so only 1 core was able to used for the computations; the CPU type was an Intel® Xeon® E5503 dual core processor with a maximum TDP of 80 W. Given that this is a dual core processor and only one core is being used for the computation, it was assumed that the TDP for a single core was half the listed amount, so a value of 40 W was used for these comparisons. For the GPUs, 3 Nvidia Tesla C2075's and 1 Nvidia Tesla C2050 used the OpenCL integration implementation and these GPUs have a combined maximum thermal design power of 913 W. The GPUs showed a factor of speedup of 561 over the single core CPU and a factor 25 better in terms of $\smile$. Even though there was much more power being supplied to the GPUs, the

comparison their  shows that the GPU implementation is better by about a factor of 25 for doing these computations versus the CPU.

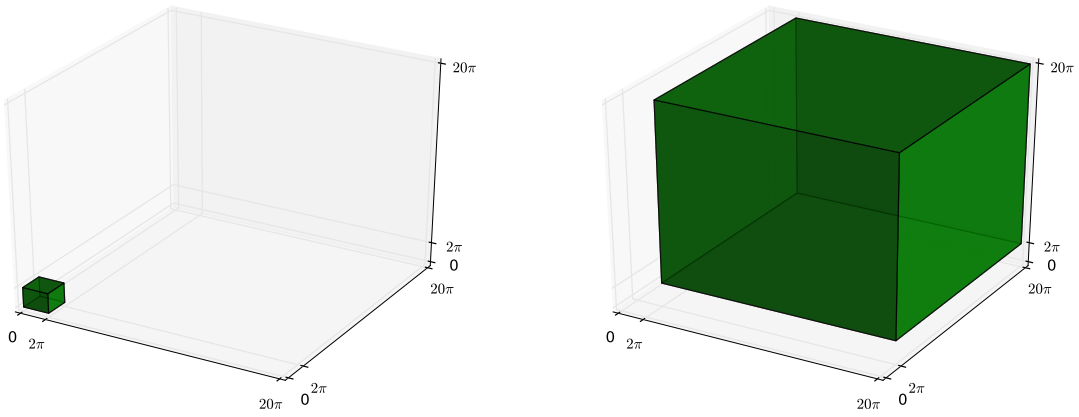

Figure 4.1: For the 3D case, first two regions of integration are shown, and notice that the upper limits of integration for the smaller region are the lower limits of integration for the larger region.

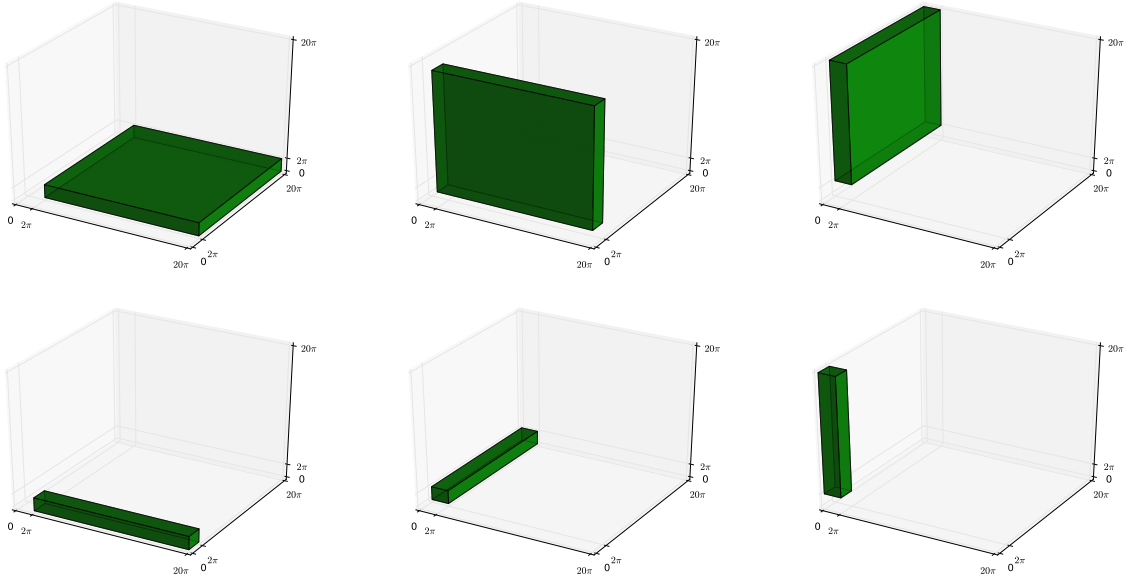

Figure 4.2: Integration of the remaining 6 hyper rectangular regions to fill the space missed by integration over the first 2 regions.

5. SEMI-ANALYTIC SOLUTION RESULTS

Semi-analytic solutions results are presented for 2 different problems. The first problem is a 3 temperature problem with a 1-D slab radiation source and the second problem is a 2 temperature problem with a 1-D spherical radiation source.

5.1 1-D Slab Radiation Source

5.1.1 2 Temperature Problem

As a method of verification, a comparison to the analytic solutions obtained in [1] was completed. All this required was run the problems with material property coefficients of 0, or $\hat{\gamma} = \kappa_e = 0$. The results of these calculations are shown in tables 5.1 - 5.2 and are converged to 5 decimal places, 1 decimal place better than claimed by [1].

5.1.2 3 Temperature Problem

For this problem, the material property coefficients were chosen to match [2], thus $\hat{\gamma} = 1/2$ and $\kappa_e = 1/6$. The initial and boundary conditions are

$$\mathbf{u} = \left[\frac{E_r}{aT_H^4}, \frac{T_e^4}{T_H^4}, \frac{T_i^4}{T_H^4} \right], \tag{5.1}$$

$$\lim_{x \to \pm\infty} \mathbf{u}\left(x, t\right) = 0, \tag{5.2}$$

$$\mathbf{u}\left(x, 0\right) = 0. \tag{5.3}$$

In addition to these conditions, the radiation source must be defined. The radiation source was defined to be

$$S_r(z,\tau) = \begin{cases} \frac{1}{2}\left[\frac{1}{2z_0}\left(\Theta(z+z_0) - \Theta(z-z_0)\right)\right] & ,0 \le \tau \le \tau_0 \\ 0 & ,\text{ otherwise} \end{cases} \tag{5.4}$$

where $\Theta(z)$ is the Heaviside step function $\Theta(z) = \int_{-\infty}^{z} \delta(s)\,\mathrm{d}s$ where δ is the Dirac delta function. The solutions are shown in tables 5.3 - 5.5 and selected output in figure 5.1. The solution (all three components) is plotted individually for every time in Figures 5.2 - 5.4.

5.2 1-D Spherical Radiation Source

For this problem, the material property strength factors were $\hat{\gamma} = \kappa_e = 0$ to recover the 2 temperature physics. The initial and boundary conditions are

$$\mathbf{u} = \left[\frac{E_r}{aT_H^4}, \frac{T^4}{T_H^4}\right], \tag{5.5}$$

$$\lim_{x\to\pm\infty} \mathbf{u}(x,t) = 0, \tag{5.6}$$

$$\mathbf{u}(x,0) = 0. \tag{5.7}$$

In addition to these conditions, the radiation source must be defined. For this problem, the Green's functions for these equations were used. This means that the radiation source was assumed to be a plane source that was pulsed at $\tau = 0$. Given that the equations are linear, the solution to a point source can be convolved with a spatial and temporal shape function on a spherical domain to obtain the solution

for a spherical radiation source. The radiation source initially considered is

$$S_r(z,\tau) = \delta(z)\,\delta(\tau)\,. \tag{5.8}$$

In 1-D, this source is more analogous to a planar source rather than a point source, but the solution for a point source is sought, so the plane-to-point transform is used. The solution for the above radiation source will be called $\mathbf{u}_{planar}$. The progression form planar to point to spherical shell to spherical solution can be seen in [2]. The result of this math is that the solution for a spherical source can be written in terms of the planar solution as follows

$$\mathbf{u}_{spherical}(r,\tau) = \int_0^A \frac{a}{r}\left(\mathbf{u}_{planar}(|r-a|,\tau) - \mathbf{u}_{planar}(|r+a|,\tau)\right)\mathrm{d}a, \tag{5.9}$$

where A is the radius of the spherical source. Due to the greater amount of accumulated error, tables 5.6 - 5.7 are truncated after the fourth number to the right of the decimal point, to be conservative. Solutions can also be seen in figures 5.5 - 5.7.

Table 5.1: Radiation energy density with $\hat{\gamma} = \kappa_e = 0$

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.09532	0.27529	0.64315	1.20069	2.23582	0.69019	0.35720
0.10000	0.09532	0.27529	0.63594	1.18872	2.21955	0.68974	0.35714
0.17783	0.09532	0.27529	0.61963	1.16204	2.18356	0.68877	0.35701
0.31623	0.09532	0.26272	0.56190	1.07186	2.06453	0.68572	0.35661
0.45000	0.08824	0.20313	0.44711	0.90953	1.86076	0.68117	0.35602
0.50000	0.04766	0.13765	0.35808	0.79903	1.73182	0.67907	0.35574
0.56234	0.00376	0.06278	0.25372	0.66680	1.57496	0.67616	0.35536
0.75000		0.00279	0.11432	0.44675	1.27399	0.66546	0.35393
1.00000			0.03647	0.27540	0.98782	0.64692	0.35141
1.33352			0.00289	0.14531	0.70822	0.61538	0.34697
1.77828				0.05967	0.45016	0.56351	0.33922
3.16228				0.00116	0.09645	0.36966	0.30347
5.62341					0.00363	0.10831	0.21382
10.00000						0.00391	0.07206
17.78279							0.00272

Table 5.2: Material energy density with $\hat{\gamma} = \kappa_e = 0$

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.00468	0.04093	0.27131	0.94687	2.11192	0.70499	0.35914
0.10000	0.00468	0.04093	0.26869	0.93715	2.09597	0.70452	0.35908
0.17783	0.00468	0.04093	0.26264	0.91540	2.06065	0.70348	0.35895
0.31623	0.00468	0.04034	0.23982	0.84093	1.94371	0.70020	0.35855
0.45000	0.00455	0.03314	0.18826	0.70288	1.74297	0.69532	0.35794
0.50000	0.00234	0.02046	0.14192	0.60493	1.61539	0.69307	0.35766
0.56234	0.00005	0.00635	0.08838	0.48846	1.46039	0.68996	0.35727
0.75000		0.00006	0.03014	0.30656	1.16591	0.67850	0.35582
1.00000			0.00625	0.17519	0.88991	0.65869	0.35326
1.33352			0.00016	0.08352	0.62521	0.62507	0.34875
1.77828				0.02935	0.38688	0.57003	0.34087
3.16228				0.00018	0.07615	0.36727	0.30456
5.62341					0.00241	0.10311	0.21377
10.00000						0.00343	0.07123
17.78279							0.00261

Table 5.3: 3 Temperature radiation energy density results with $\hat{\gamma} = 1/2$ and $\kappa_e = 1/6$

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.09532	0.27510	0.62610	1.00736	1.60279	0.44486	0.23613
0.10000	0.09532	0.27507	0.61893	0.99839	1.59234	0.44458	0.23609
0.17783	0.09532	0.27499	0.60279	0.97837	1.56914	0.44399	0.23601
0.31623	0.09531	0.26207	0.54647	0.91043	1.49127	0.44213	0.23575
0.45000	0.08820	0.20247	0.43814	0.78491	1.35132	0.43936	0.23536
0.50000	0.04766	0.13757	0.35363	0.69295	1.25285	0.43809	0.23518
0.56234	0.00379	0.06335	0.25458	0.58359	1.13460	0.43631	0.23493
0.75000		0.00309	0.11844	0.41177	0.93143	0.42979	0.23399
1.00000		0.00002	0.03910	0.27276	0.74264	0.41849	0.23235
1.33352			0.00353	0.15664	0.55282	0.39922	0.22945
1.77828			0.00004	0.07030	0.36859	0.36750	0.22439
3.16228				0.00158	0.09088	0.24736	0.20103
5.62341					0.00431	0.07836	0.14236
10.00000						0.00346	0.04894
17.78279							0.00200

Table 5.4: 3 Temperature electron energy density results with $\hat{\gamma} = 1/2$ and $\kappa_e = 1/6$

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.00461	0.03823	0.20339	0.55874	1.18916	0.45403	0.23740
0.10000	0.00460	0.03788	0.20010	0.55304	1.18154	0.45374	0.23736
0.17783	0.00459	0.03697	0.19292	0.54065	1.16500	0.45310	0.23728
0.31623	0.00440	0.03284	0.17030	0.50207	1.11352	0.45111	0.23701
0.45000	0.00328	0.02401	0.13743	0.44666	1.03958	0.44814	0.23662
0.50000	0.00230	0.01949	0.12310	0.42243	1.00706	0.44677	0.23644
0.56234	0.00113	0.01395	0.10514	0.39122	0.96459	0.44487	0.23618
0.75000	0.00007	0.00363	0.05951	0.30159	0.83593	0.43789	0.23523
1.00000		0.00038	0.02289	0.20391	0.67825	0.42581	0.23356
1.33352		0.00001	0.00426	0.11399	0.50327	0.40529	0.23062
1.77828			0.00026	0.04747	0.32971	0.37165	0.22547
3.16228				0.00085	0.07585	0.24610	0.20175
5.62341					0.00315	0.07511	0.14233
10.00000						0.00309	0.04840
17.78279							0.00192

Table 5.5: 3 Temperature ion energy density results with $\hat{\gamma} = 1/2$ and $\kappa_e = 1/6$

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.00008	0.00208	0.03773	0.29675	1.02242	0.47428	0.24001
0.10000	0.00008	0.00206	0.03713	0.29325	1.01526	0.47394	0.23997
0.17783	0.00008	0.00203	0.03578	0.28563	0.99970	0.47320	0.23988
0.31623	0.00008	0.00185	0.03135	0.26178	0.95131	0.47087	0.23961
0.45000	0.00006	0.00135	0.02444	0.22728	0.88177	0.46742	0.23920
0.50000	0.00004	0.00105	0.02130	0.21217	0.85122	0.46582	0.23901
0.56234	0.00002	0.00069	0.01741	0.19287	0.81143	0.46362	0.23875
0.75000		0.00014	0.00840	0.13948	0.69199	0.45553	0.23777
1.00000		0.00001	0.00255	0.08573	0.54839	0.44160	0.23604
1.33352			0.00036	0.04169	0.39364	0.41811	0.23299
1.77828			0.00002	0.01410	0.24634	0.38001	0.22768
3.16228				0.00012	0.04885	0.24267	0.20321
5.62341					0.00150	0.06834	0.14225
10.00000						0.00242	0.04729
17.78279							0.00177

Table 5.6: Radiation Energy Density for 2-T spherical radiation source with $A = 0.75$.

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.1907	0.5507	1.2162	1.7439	2.2122	0.0502	0.0066
0.10000	0.1906	0.5506	1.2090	1.7316	2.1955	0.0501	0.0066
0.17783	0.1906	0.5506	1.1935	1.7060	2.1639	0.0501	0.0066
0.31623	0.1906	0.5506	1.1402	1.6225	2.0617	0.0498	0.0065
0.45000	0.1906	0.5501	1.0461	1.4893	1.9014	0.0494	0.0065
0.50000	0.1906	0.5420	0.9986	1.4234	1.8235	0.0493	0.0065
0.56234	0.1906	0.5175	0.9271	1.3262	1.7098	0.0490	0.0065
0.75000	0.0921	0.2479	0.5190	0.8217	1.1497	0.0481	0.0065
1.00000		0.0041	0.1261	0.3158	0.5712	0.0466	0.0064
1.33352			0.0216	0.1249	0.3039	0.0441	0.0064
1.77828				0.0411	0.1497	0.0399	0.0062

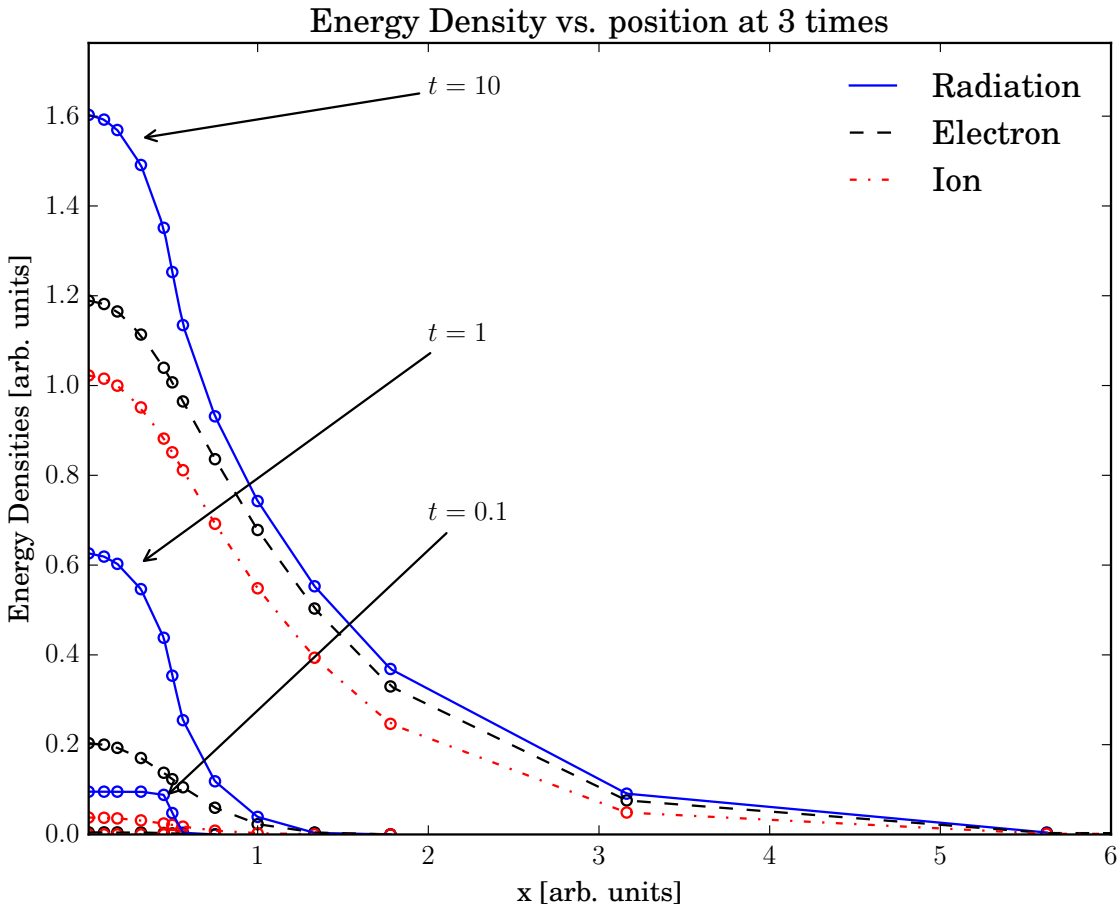

Figure 5.1: The non-dimensional radiation, electron, and ion energy densities are plotted above for 3 different times. Results show that all energy densities are monotonically increasing when $t \leq 10$, which is when the source drops down to 0. Energy is deposited into the radiation field from the source, which then is transferred to the electron field, and finally transferred to the ion field. This result is clearly seen as the ion energy density is less than the electron energy density which is less than the radiation energy density. Values actually computed are circled and linearly interpolated in between.

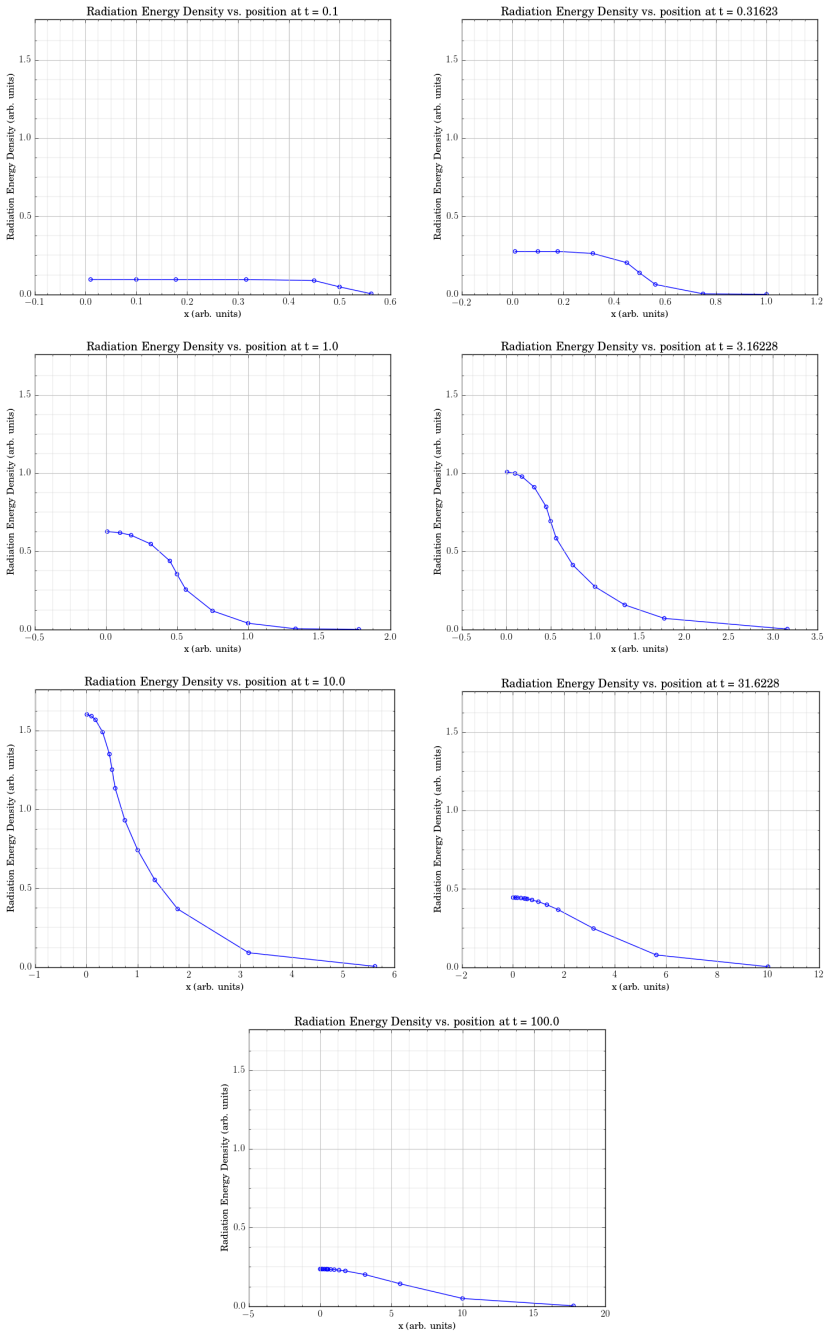

Figure 5.2: The radiation energy density is plotted as a function of position for at times coincident with those presented in [1]. The radiation energy density is monotonically increasing while the source is on ($t \leq 10$). After the source is turned off, the energy spreads throughout the domain, decreasing near $x = 0$ and increasing at larger x. Values actually computed are circled and linearly interpolated in between.

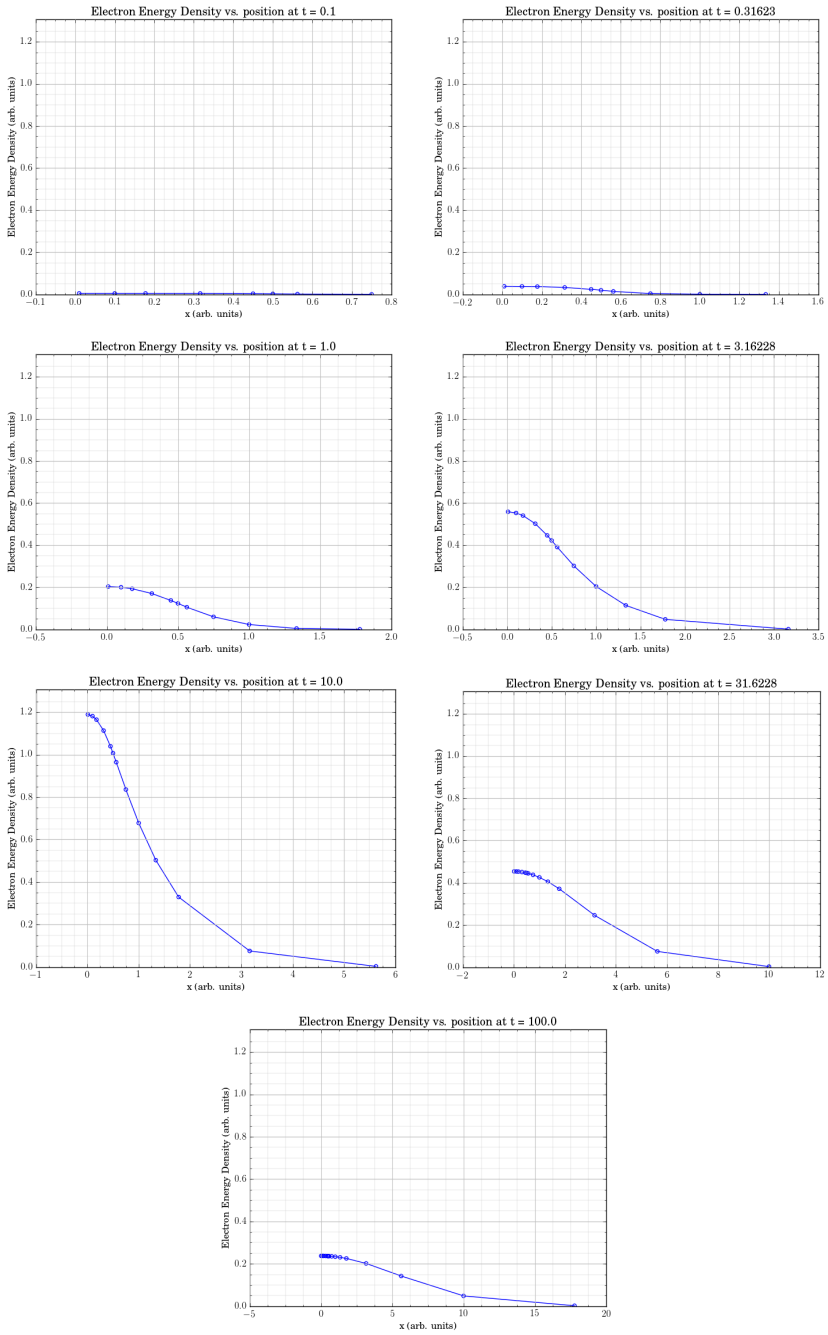

Figure 5.3: The electron energy density is plotted as a function of position for at times coincident with those presented in [1]. The electron energy density is monotonically increasing while the source is on ($t \leq 10$). After the source is turned off, the energy spreads throughout the domain, decreasing near $x = 0$ and increasing at larger x. Values actually computed are circled and linearly interpolated in between.

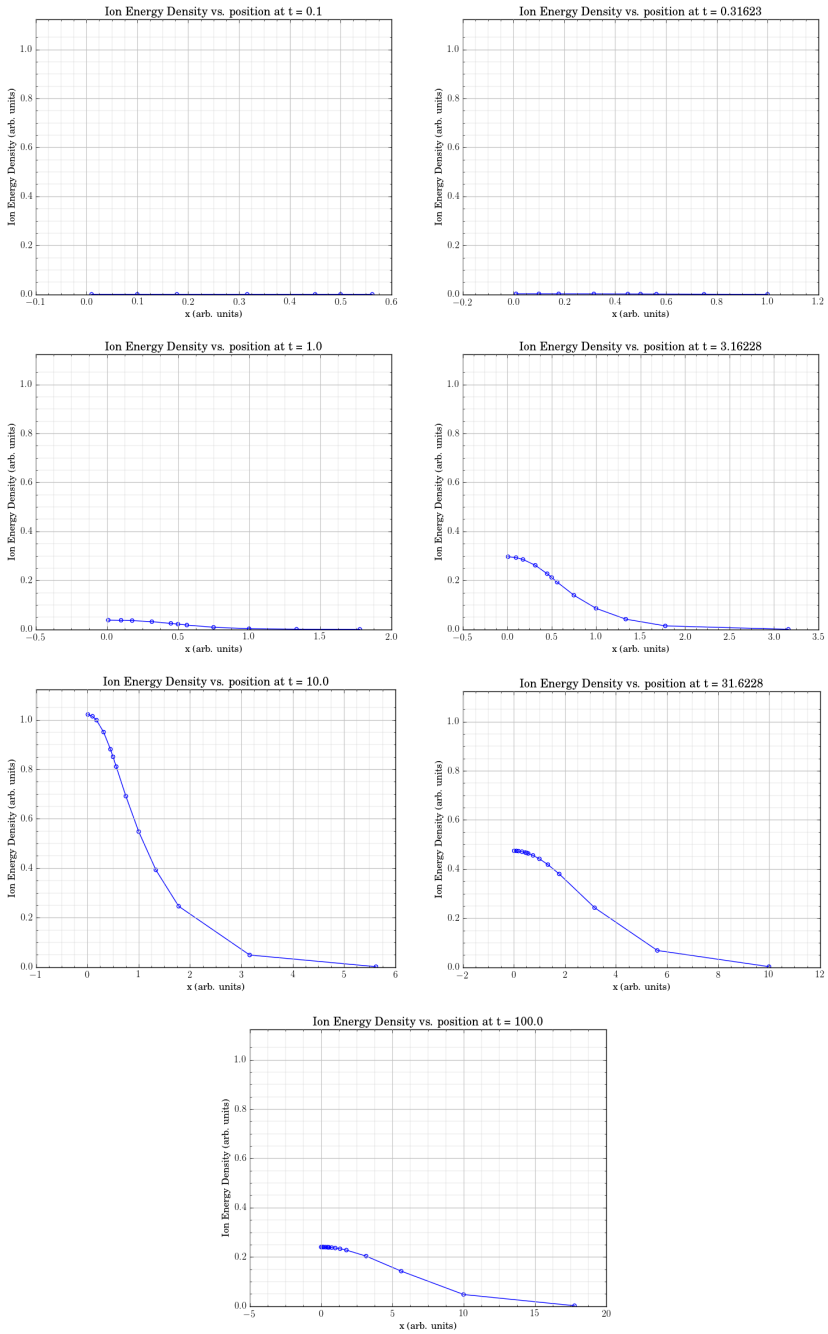

Figure 5.4: The ion energy density is plotted as a function of position for at times coincident with those presented in [1]. The ion energy density is monotonically increasing while the source is on ($t \leq 10$). After the source is turned off, the energy spreads throughout the domain, decreasing near $x = 0$ and increasing at larger x. Values actually computed are circled and linearly interpolated in between.

Table 5.7: Material Energy Density for 2-T spherical radiation source with $A = 0.75$.

$x\backslash t$	0.10000	0.31623	1.00000	3.16228	10.00000	31.62280	100.00000
0.01000	0.0094	0.0819	0.5421	1.4762	2.1759	0.0538	0.0067
0.10000	0.0094	0.0819	0.5395	1.4667	2.1616	0.0538	0.0067
0.17783	0.0094	0.0819	0.5335	1.4457	2.1303	0.0537	0.0067
0.31623	0.0094	0.0819	0.5131	1.3767	2.0289	0.0534	0.0067
0.45000	0.0094	0.0819	0.4777	1.2652	1.8699	0.0530	0.0066
0.50000	0.0094	0.0817	0.4588	1.2095	1.7926	0.0528	0.0066
0.56234	0.0094	0.0804	0.4290	1.1265	1.6797	0.0525	0.0066
0.75000	0.0046	0.0381	0.2263	0.6797	1.1221	0.0515	0.0066
1.00000			0.0360	0.2366	0.5473	0.0498	0.0066
1.33352			0.0032	0.0848	0.2850	0.0469	0.0065
1.77828				0.0245	0.1363	0.0422	0.0063
3.16228				0.0003	0.0173	0.0258	0.0056
5.62341					0.0004	0.0065	0.0039
10.00000						0.0002	0.0013

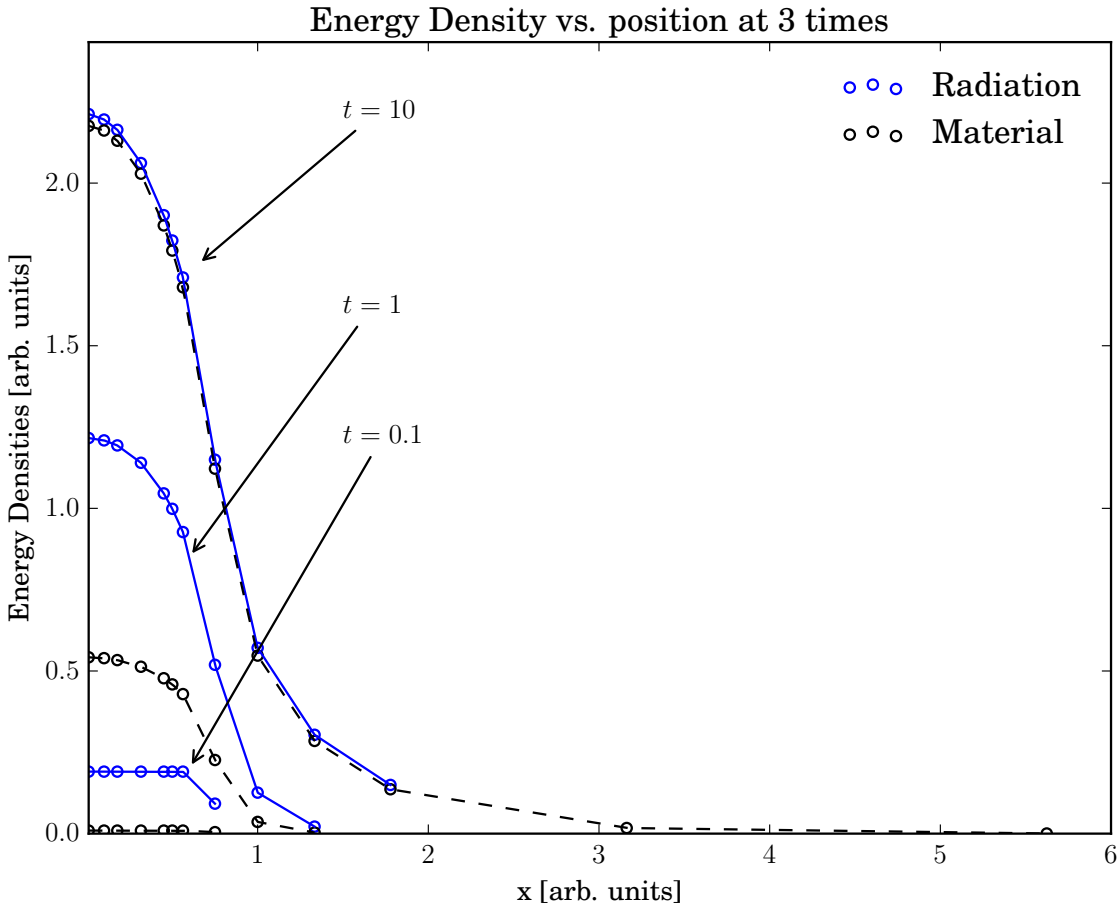

Figure 5.5: The non-dimensional radiation and material energy densities are plotted above for 3 different times. Results show that the energy densities are monotonically increasing when $t \leq 10$, which is when the source drops down to 0. Energy is deposited into the radiation field from the source, which then is transferred to the material field. This result is clearly seen as the material energy density is less than the radiation energy density. Values actually computed are circled and linearly interpolated in between.

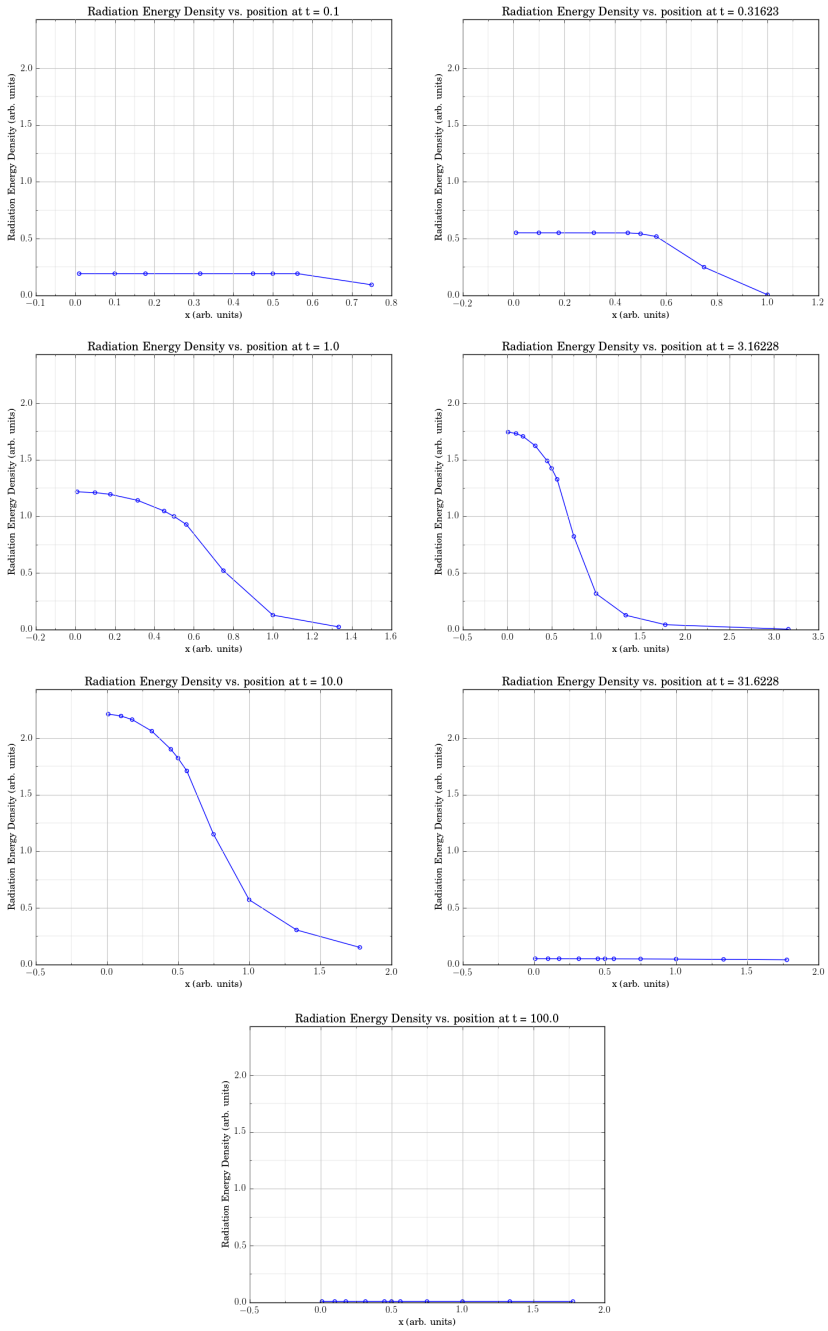

Figure 5.6: The radiation energy density is plotted as a function of position for at times coincident with those presented in [1]. The radiation energy density is monotonically increasing while the source is on ($t \leq 10$). After the source is turned off, the energy spreads throughout the domain, decreasing near $x = 0$ and increasing at larger x. Values actually computed are circled and linearly interpolated in between.

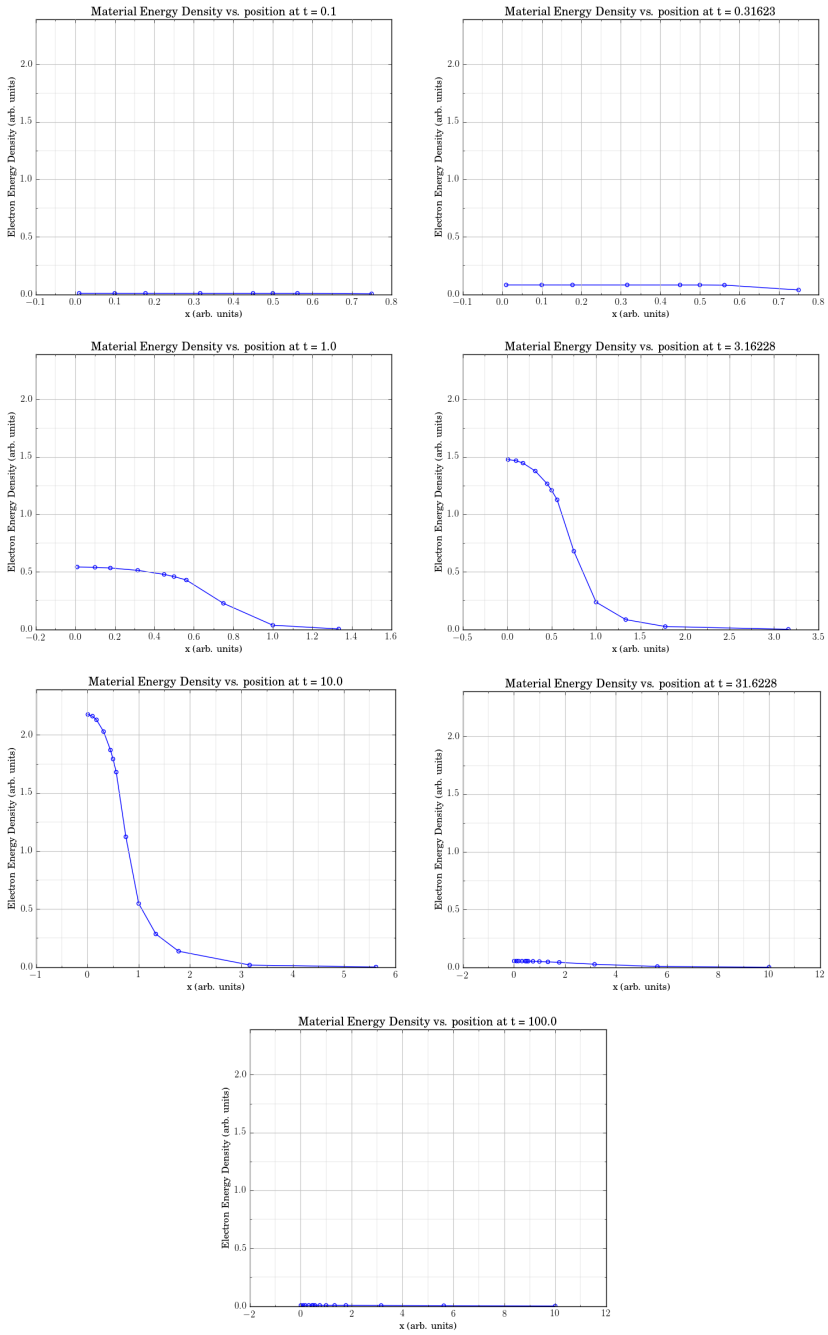

Figure 5.7: The material energy density is plotted as a function of position for at times coincident with those presented in [1]. The material energy density is monotonically increasing while the source is on ($t \leq 10$). After the source is turned off, the energy spreads throughout the domain, decreasing near $x = 0$ and increasing at larger x. Values actually computed are circled and linearly interpolated in between.

6. CONCLUSIONS

The analytic solutions presented can be used for verification of codes that simulate physical systems that involve thermal radiative transfer. More specifically, benchmark solutions were obtained for two different material models, one involving a single material species, and the other in which the electrons and ions within a material are treated separately do not have to be in thermal equilibrium with each other. These models are the 2-T and 3-T models, respectively, and while the material models were different, the radiation energy was modeled using full transport for all of the problems considered. This is in contrast to a radiative diffusion model, which is less physically accurate. The problems solved used only external radiation energy sources that were constant over a finite spatial and temporal extent. The 2-T model assumed a spherical radiation source which could in principle test 1-D radial (r), 2-D radial-inclination angle ($r,\ \theta$), or 3-D (any coordinate system) codes. The 3-T model assumed a 1-D slab radiation source that is useful for testing out codes capable of solving the equations on 1-D Cartesian meshes.

The problems were solved via non-dimensionalization and linearization of the coupled equations in terms of the nondimentional scalar intensity and material temperature(s) raised to the 4th power. Spatial Fourier and temporal Laplace transforms were applied and the linear system of equations was solved, followed by the inverse transforms. The composite Gauss-Legendre numerical integration scheme was used to compute the inverse transforms. Multiple techniques were used to decrease the time to solution. First, numintCL was written to run on multiple GPUs simultaneously through OpenCL, which allowed for a factor of speedup of ~ 561 over a single core CPU implementation of the same algorithm. A power normalized figure of merit,

, was introduced to correct for the power consumption differences and the GPU implementation outperformed the CPU implementation by a factor of ~ 25. In addition to hardware acceleration, multiple algorithmic acceleration techniques were applied including sequence acceleration, asymmetric refinement, and error based de-refinement. When the devices are operating at peak capacity, a refinement of the composite rule exponentially increases the compute time, and thus the importance of the algorithmic acceleration techniques cannot be overstated. The combination of advanced hardware and application of accelerated algorithms were both required in order for this work to be completed in reasonably timed manner.

Throughout the course of this work, other important information was gleaned as well as lessons learned. One such lesson is the importance of proper software design in terms of writing general code. Although highly important to keep in mind writing software in the most general and extensible way, the ultimate goals of a given piece of code need to be clearly outlined, so that valuable development time is not used generalizing the code in ways that in will likely never be used. In addition, it is very clear that much less effort will be spent in the software development process when incremental and well tested changes are implemented as opposed to large and sweeping changes. Although sometimes change is unavoidable, tasks should almost always be broken up into their smallest constituents and tested both individually and as a whole before a given functionality is added. This is not only a good model for software development, but for most forms of scientific development as well.

REFERENCES

[1] B. Su, G. L. Olson, An analytical benchmark for non-equilibrium radiative transfer in an isotropically scattering medium, Annals of Nuclear Energy 24 (13) (1997) 1035–1055.

[2] R. G. McClarren, J. G. Wöhlbier, Solutions for ion-electron-radiation coupling with radiation and electron diffusion, Journal of Quantitative Spectroscopy and Radiative Transfer 112 (1) (2011) 119–130.

[3] R. G. McClarren, D. A. Holladay, Electron-ion-radiation coupling benchmarks for verification of hedp/ife codes, Fusion Science and Technology 60 (2) (2011) 600–604.

[4] E. J. Weniger, Prediction properties of aitken's iterated δ^2 process, of wynn's epsilon algorithm, and of brezinski's iterated theta algorithm, Journal of Computational and Applied Mathematics 122 (2000) 329–356.

[5] A. J. Macleod, Acceleration of vector sequences by multi-dimensional δ^2 methods, Communications in Applied Numerical Methods 2 (1986) 385–392.

[6] P. Wynn, On a device for computing the $e_m(s_n)$ transformation, Mathematical Tables and Other Aids to Computation 10 (54) (1956) 91–96.

[7] J. Wimp, Sequence Transformations and Their Applications, Vol. 154 of Mathematics in Science and Engineering, Academic Press, Inc., 1981.

[8] B. Catanzaro, Opencl™ optimization case study: Simple reductions (August 2010) [cited 10/12/14].
URL `http://developer.amd.com/resources/documentation-articles/articles-whitepapers/opencl-optimization-case-study-simple-reductions/`